Return to July 24, 2000 table of contents

Access To Instruments Will Change

For the test and measurement world, as instruments begin to embed Win32 operating systems, the way instruments are accessed can be revolutionized by DCOM. For instance, the Agilent Infinium oscilloscope currently uses Windows 98 as its embedded operating system. In the above example, the client made DCOM calls to the remote server object on another computer, which then communicated to the instrument with SCPI over a local I/O transport. In an embedded Win32 environment, the server object could have been hosted inside the instrument directly interfacing to the firmware. The client connects to this programmatic server interface with DCOM to the instrument over a network.

Previously broached was the fact that combining the interfaces with IntelliSense improves the programming model. Another approach to simplifying the model is to layer an ActiveX control object on top of the instrument server to automate some of the programming tasks by allowing the client to modify values through the use of GUI controls. Microsoft currently uses this approach when layering a timer control on top of its timer API.

Listing 5

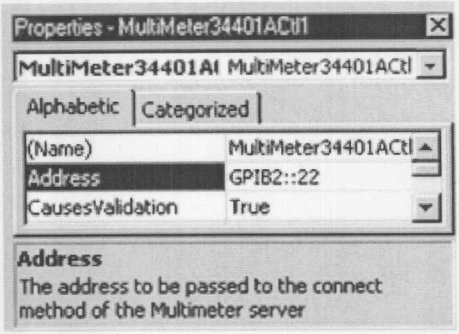
```
Public Property Get Address() As String
Address = m_address
End Property
Public Property Let Address(ByVal newAddress As String)

m_address = newAddress
Set m_multimeter = Nothing
If Not m_address = "" Then
Set m_multimeter = New MultiMeter34401A
m_multimeter.Connect m_address
End If
End Property
```

This particular example consists of layering an ActiveX control on top of the multimeter instrument server to provide support for triggering. The programmer begins with the usual course of starting with the VB development tool, and when the new project dialog appears, the programmer selects "*ActiveX Control*." Once again, the programmer changes the default supplied VB development tool project name and the "user control" identifier to "MultimeterCtl" and "Multimeter34401Actl," respectively. Next comes the addition of a property called "Address" to aid the user in connecting to the multimeter. The definitions will look like ***Listing 5 (above).***

In addition, we will add controls to the form that allow the user to configure, start, stop, and display a trigger. The control and the configure dialog are shown in *Figure 4*.

Note that the definition of the range and resolution arguments is improved. In the server API, these arguments were defined as the Variant type. Assurances were made that the user passed in one of the expected strings of "MAX," "MIN," and "DEF." If this example had truly been production code, verification that the argument wasn't of other Variant values, like dates and currency, etc., would have occurred. In the dialog, however, the user can only select one of the predefined string values, or enter a numeric value.

Using the control in a client application, such as a form, entails adding it to the project's components by selecting the project menu dropdown, the component's subitem, and the newly created multimeter control. Once added to the components, the control is taken and dropped on the form. At this point, the property browser has an item in its list for the address property that was implemented. It's set to an appropriate value ***(Fig. 5 below).***



5. As shown here, the address variable for the multimeter is set by employing the VB Property Browser.

To access the configure dialog, right-click on the control and select "Edit" from the context menu. Then, modify the appropriate settings, save the project, and run it. The form instantiates with the values that were set at design time and "start" is selected to trigger a new measurement. The result is the "programming" of a triggered measurement without adding a line of code!

Another value proposition of COM is that it's a "language-neutral" environment. This allows programmers to access the objects created in VB from other languages like C/C++, Java, and even COBOL. Rather than providing "header files" for these languages, the environments were modified to read the type library created by the VB tool that was generated from the interface definitions and placed into the executables that were deployed. For C++, a compiler directive called "import" was added to take the type library definitions and convert them to a header file representation. A sample C++ program that exercises the multimeter server can be found in *Listing 6*.

Yet, the reading of the type library isn't restricted to common programming languages. For test and measurement environments, the development tools of choice are either the VEE or LabVIEW graphical environments. Both have been modified to read COM-type libraries. A sample VEE program that accesses the multimeter server would look like *Figure 6*.

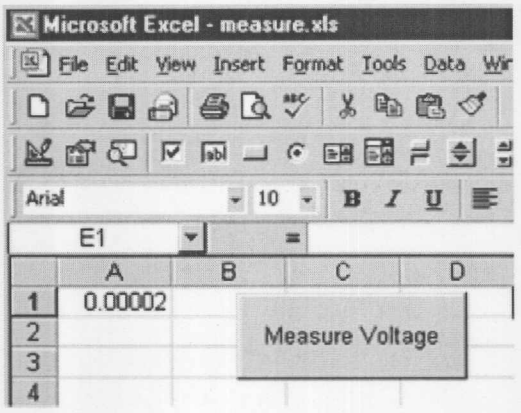
A derivative of VB, Visual Basic for Applications (VBA), has replaced the proprietary macro languages of desktop applications, such as the Microsoft Office suite and Visio. This allows powerful integration opportunities between the VB Objects that are created and the objects exposed by these desktop applications. For the multimeter example, this involves creating a button on an Excel worksheet that, when pushed, will take a measurement from the multimeter and place it in cell "A1."

To place the button on the worksheet, first activate the "Control Toolbox" by selecting the Excel view menu dropdown, followed by the "Toolbars" submenu. Then, choose the "Control Toolbox." Drag the command button on the worksheet, and change the caption to "Measure Voltage" in the properties window. Double click on the button to be taken to the body definition of the click callback in the integrated VB editor. In the tools menu dropdown, select the references' subitem, and then reference the multimeter instrument server. Add the code in **Listing 7 (below)** to the body of the callback.

Listing 7

```
Dim mm As New MultiMeter34401A
mm.Connect "GPIB3::22" ' May be different in other configurations
Application.Cells(1, 1) = mm.Measure.VoltageAC("MAX", "MAX")
```

Upon returning to the Excel spreadsheet, exit design mode, and select the newly created "Measure Voltage" button. The measurement is made and the value is placed into cell "A1" ***(Fig. 7 below).***



7. VB Objects are easily integrated with standard desktop applications. When the "Measure Voltage" button is clicked in this Excel example, the measured voltage is placed into cell "A1" of the spreadsheet.

To sum things up, COM has moved from an emerging technology to a de facto standard for component communication on personal computers. Because it's a language-independent standard, the developer can choose his or her language of implementation. Putting VB through the paces of COM development--interface design, distributed application, and integration with Microsoft and test and measurement applications--has hopefully demonstrated and confirmed that VB is a natural starting point for COM development.

Paul Faust *is a software design engineer at Agilent Technologies in the Measurement Connectivity Center of Technology, Loveland, Colo., where he's a member of the BenchLink XL development team. He's a graduate of the University of California at Santa Cruz with degrees in economics and computer science. Faust can be reached via e-mail at: paul_faust@agilent.com.*

Previous Page

Return to Top